What have I been doing on my branch?

François Michonneau

1 Cleaning up the code

I have been trying to clean up the code (removing duplicate functions, reorganizing files with many functions in sections, removing old chuncks of codes that were commented and deprecated, etc.) Have a look at `methods-phylo4.R`.

I also tried to separate code used to "build" objects and code used to test properties of objects. Indeed, this latter part should only be found in the validator. It allows us to make sure that the rules about `phylo4` objects are consistent and found in only one place.

2 Attributing internal names for labels and edges

2.1 Labels

Tip and internal node labels have now internal names that are simply the node they are supposed to document. It thus becomes possible to store labels in any order and it makes assignment of labels more robust. For the replace method, by default, the labels are attributed to the nodes in numerical order. The user can however provide a named vector (the names being the node numbers), in which case, the labels will be matched.

`checkPhylo4` now checks that labels have internal names and fails otherwise. The validator returns an error message to the user saying that the object is an old version of the object, and I created an `updatePhylo4` function which will assign automatically the right labels to the nodes (assuming that the labels are stored in correct order). I don't think we should include this function in the CRAN release. I wrote this function simply to ease the transition to this new data structure.

I wrote a function (`.createLabels`) which returns a properly formatted vector that can be used (internally) to create internal node or tip labels. This function is used by the `phylo4` constructor as well as the replace method for tip and internal node labels, which limits code redundancy.

2.2 Edges

I used a similar approach for the edges. The slots `edge.length` and `edge.labels` have also internal names that are simply the combination of the 2 nodes the

edge connects separated by a dash (e.g. 14-16).

The function `.createEdge` can create properly formatted vectors for the slots `edge.length` and `edge.labels`. This function can thus be used by both the `phylo4` constructor and the replace methods for both the lengths and the labels of the edges.

3 Rewritting phylo4d constructor

3.1 Matching data and trees

Now, data are not matched directly on node labels but are instead matched against node numbers. I did this because in the future, if we decide to allow users to use non-unique labels, it will still be possible to assign different data to different tips. (We can indeed imagine labels as being used simply for plotting purpose.) It also becomes possible to provide data directly with the node numbers. It's something that can be useful if the tree is big or if the labels are complicated names.

Matching of the data and the tree is done only when the object is returned to the user (e.g., by `tdata` or `print`). Indeed, in the slots `@node.data` and `@tip.data`, only the node numbers are stored. This is more efficient as it requires less code to update node labels: they are only stored in one place (`@tip.label` and `@node.label`).

3.2 The new functions `.phylo4Data` and `formatData`

The `phylo4d` constructor uses two new functions: `.phylo4Data` and `formatData`. `formatData` is called by `.phylo4Data` and performs all the tests to make sure that the data provided can be matched with the tree. It replaces `checkData` but is less redundant because the same code is used for `tip.data`, `node.data` and `all.data`. In addition, `formatData` takes care of matching the labels of the data with the labels in the tree. `formatData` creates a data frame with the correct dimensions given the tree. The data frame is then populated with the data provided. Doing things this way is more robust to missing data and/or not-properly formatted data. `formatData` returns properly formatted data as a list of two elements (`$tip.data` and `$node.data`).

`.phylo4Data` then takes the objects returned by `formatData` to bind the data frames together if more than just `tip.data`, `node.data` or `all.data` are provided.

The main advantage of using these functions is that they can be called by other functions in `phylobase` such as `tdata<-` and the new `addData`.

These two functions deprecate `checkData` and `attachData`.

3.3 Tests

I wrote a series of tests using `RUnit` to make sure that all the options and the new work properly.

4 Other changes

4.1 New functions

- `addData`: this function allows the user to append data to an existing `phylo4d` object.
- `edgeLength<-`: this function allows the user to supply or update edge lengths.

4.2 Modified functions

- `tdata`: if the tree doesn't have node labels, node numbers are returned instead of empty values. A new option (`empty.colums`) allows the user to not return columns filled with `NA`. This can happen for instance if the user asks for tip data and for some measurements (i.e. columns) there are only data associated with internal nodes.
- `getEdge`: this function can return all the edges in a tree, or for a set of edges, it can return the other extremity of the edges or the full edge.
- `checkPhylo4`: now checks for the correct lengths of `edge.length` and `edge.labels`, and makes sure that the values of `edge.length` are of class `numeric`.
- `getNode`: updated to reflect changes in the way elements in objects are stored and the node matching process is now independant of the order in which the labels are stored.
- `hasNodeLabels`: test changed to reflect the way empty node labels are stored.
- `as(''phylo4d'', ''data.frame'')`: updated to reflect changes in the way data are stored in `phylo4d` objects
- `prune`: updated to make it more robust and to accomodate internal labels